INDIGO - DataCloud

SaaS via a Portal: FutureGateway

Riccardo Bruno INFN

riccardo.bruno@ct.infn.it

Marco Fargetta INFN

marco.fargetta@ct.infn.it

INDIGO-DataCloud is co-founded by the Horizon 2020Framework Programme

FutureGateway

Outline

- Introduction
- Implementation
- ToscaIDC EI
- Example
- New applications

Introduction

Motivations

New research requires an ever increasing amount of both computational power and storage space

- Distributed Computing Infrastructures (DCIs) are a concrete answer to solve this need
- Unfortunately the use of DCIs requires a strong technical background
- Different DCIs use different background technologies
- The introduction of the the Science Gateway concept currently provides a successful solution

FutureGateway

A framework to build **Science Gateways:**

- "A Science Gateway is a community-developed set of tools, applications, and data that is integrated via a portal or a suite of applications, usually in a graphical user interface, that is further customized to meet the needs of a specific community." TeraGrid-XSEDE
  - Serves Community
  - Tools applications and data integration
  - Provides GUI (Web, desktop and mobile applications)
  - Customization

Design principles

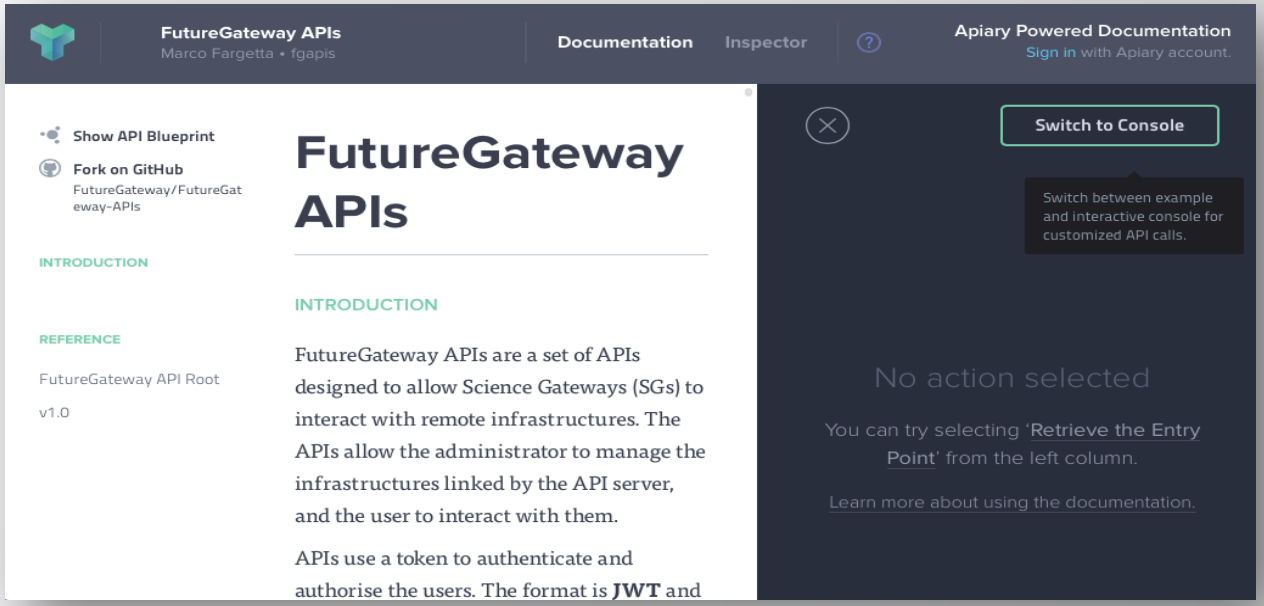
- Ease the installation and the maintenance
  - It provides installation and maintenance scripts
  - All sources available on GitHub
  - Easily customizable for different community needs
- Flexible and structured access to distributed computing services through pluggable modules
  - The physical access to the distributed infrastructures comes through the JSAGA library (Grid&Cloud Engine)
  - PaaS access through the TOSCA orchestrator (ToscaIDC)
- Provide restFull APIs
  - Back-end portal independency.
  - Mobile and desktop applications.
  - Accessible by any programming language.

Concepts

- FG manages three entities:
  - **Applications**
    - It describes the activity to perform on the DCI.
  - **Infrastructures**
    - Describe the DCI environment where the application can run. It contains the necessary information to allow the application to physically access DCI resources.
    - Each application may execute on one or more infrastructures.
  - **Tasks**
    - Applications executing on a DCI are tasks. The term 'task' may include many operations ranging from simple batch executions, up to more sophisticated actions like a PaaS creation and exploitation

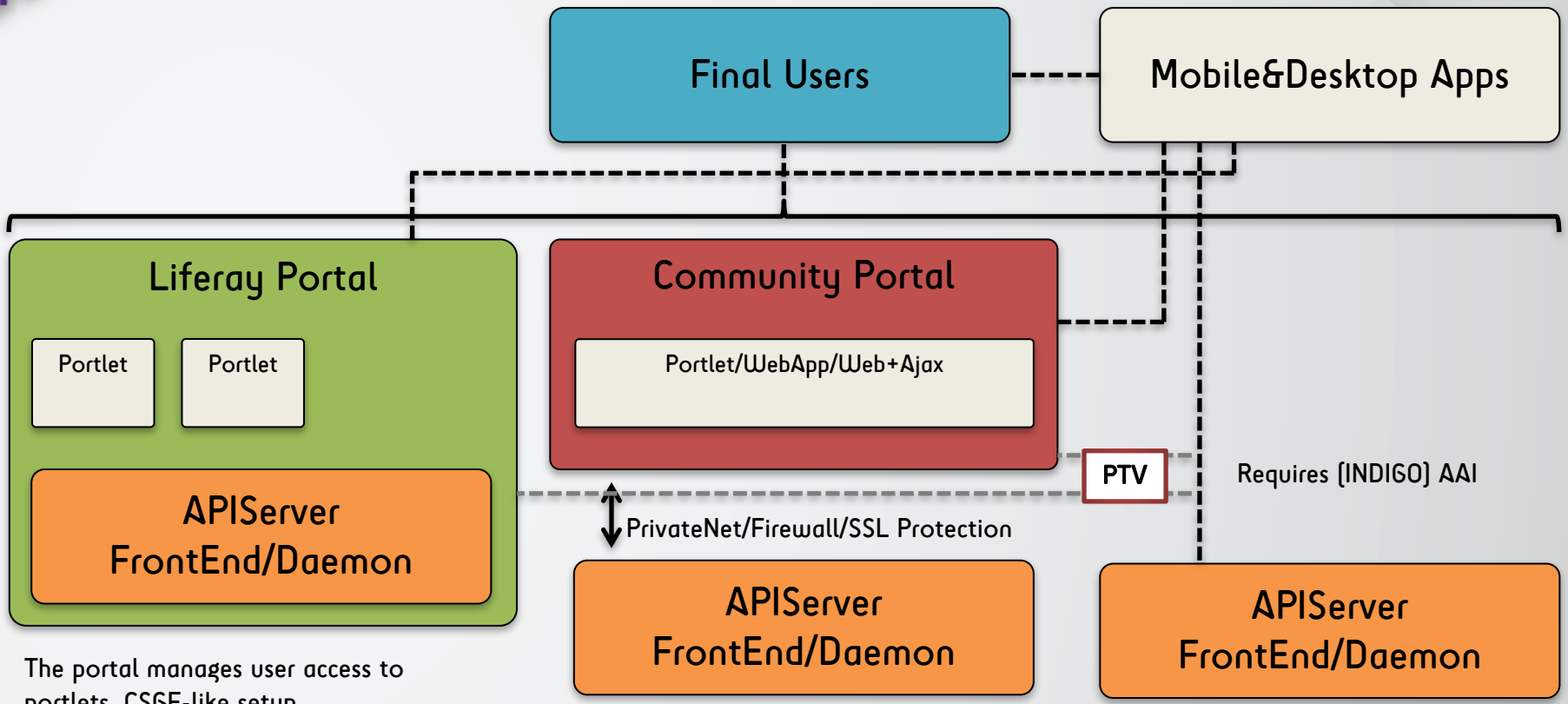
API Specifications

- Available at: http://docs.fgapis.apiary.io/#



- Three main endpoints:
  - Tasks
  - Applications
  - Infrastructures

Usage scenarios



The portal manages user access to portlets. CSGF-like setup

Used by:

- INDIGO SG Demonstrator
  https://fgw02.ncg.ingrid.pt

Used by:

- Molecular Dynamics
- Climate change (Kepler)

Used by:

Galaxy portlet
Desktop
Mobile apps (Under test)

Components

FutureGateway Database (FGDB)

- Keeps and maintains:
  - Tasks, Applications, Infrastructures, task-queue, users, groups and roles

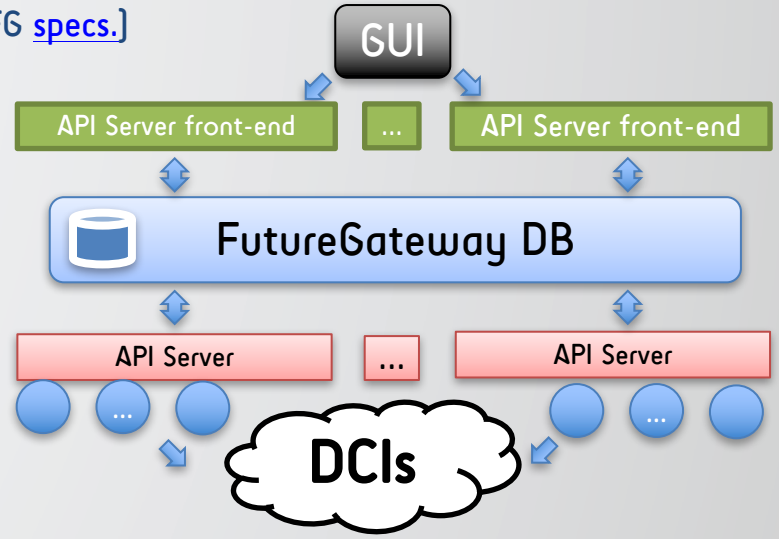
API Server Front-end

- Accepts API calls in accordance with the defined specifications
- Fill-up a queue table of corresponding commands (producer)
- Manage authN/Z (users/groups/roles)
- Manage Applications and Infrastructures and Tasks
- More front-ends may exist:
  - **fgAPIServer** (Actual python implementation for FG specs.)

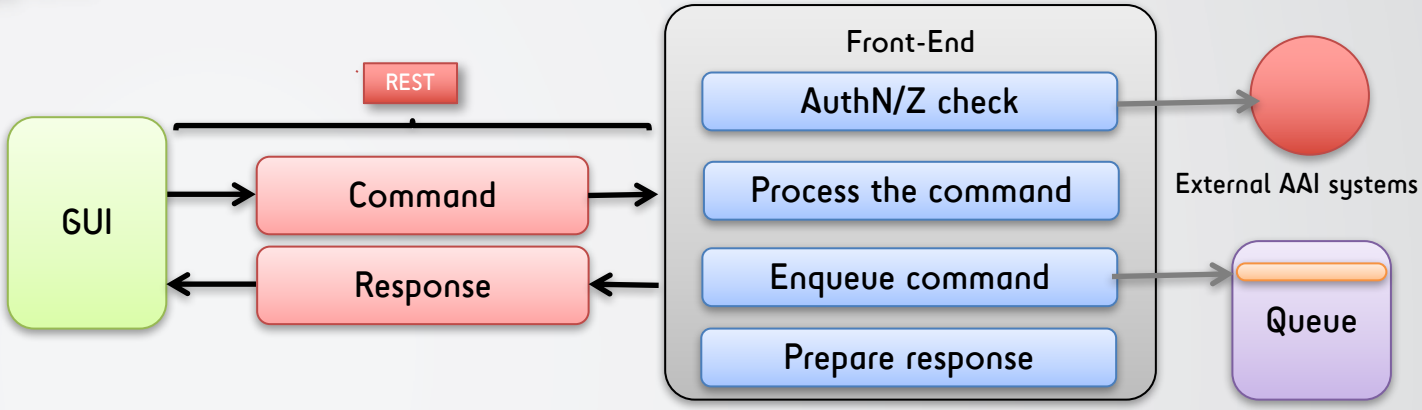
APIServer +

Executor Interfaces (EI)

- Polls over queue table (consumer)
- Extract tasks to submit and send them to the proper EI
- Check status and consistency of submitted tasks
- Retrieve available outputs (if any)
- More daemons may be developed to address any possible DCI.
  - **APIServerDaemon** (Actual java implementation)
  - Other implementation (python, ...)



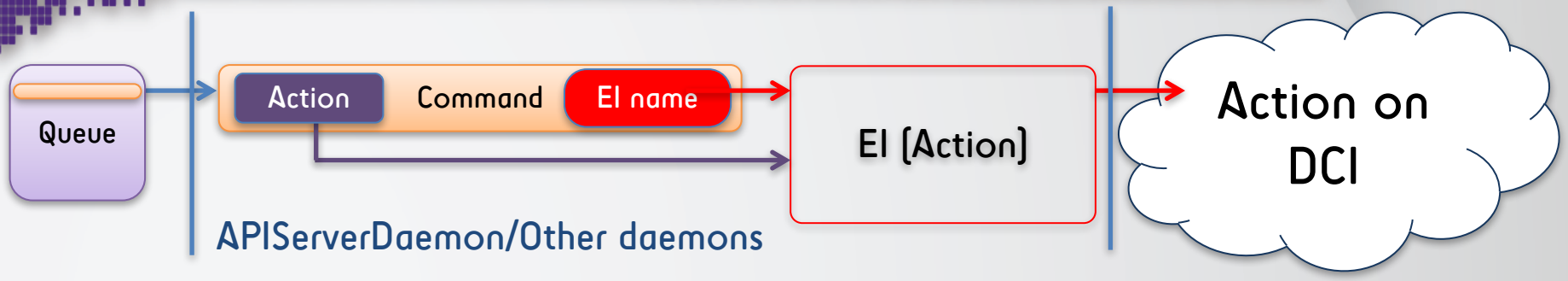
API Server Front-end



- Operations
  - GUIs send a command via REST APIs
  - Each 'command' is a JSON stream specifying the requested activity
  - The Front-End first check for requestor Authorization and Authentication eventually using external AAI mechanisms (See: Indigo-IAM and PTV service)
  - The command is processed querying and/or updating the DB accordingly
  - Commands to be finalized by the APIServer are stored in the queue table
  - Command output is returned back into the response as a JSON stream

API Server

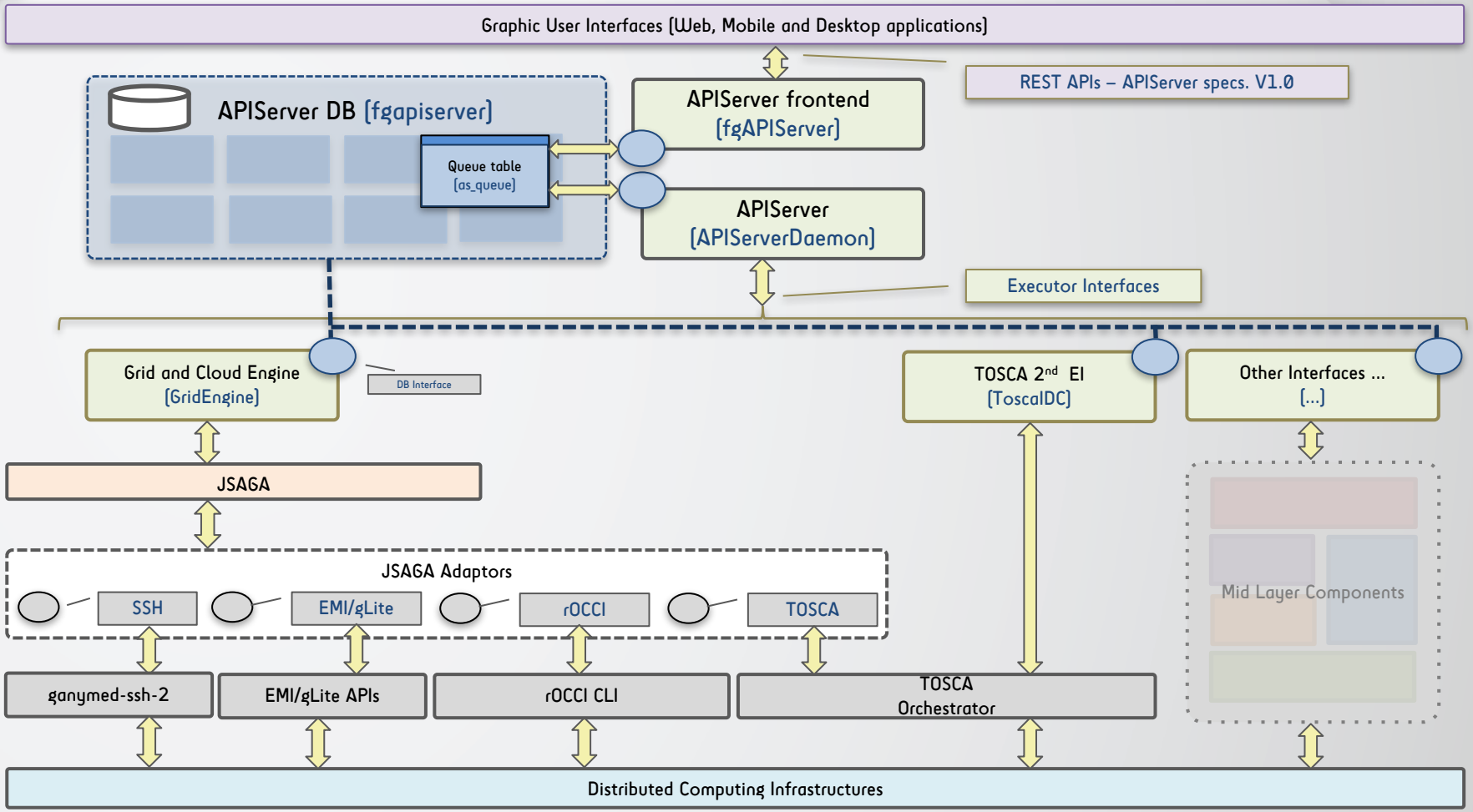
Queue Daemons and Executor Interfaces



- Operations
  - Commands (Tasks=Command(Action,EI)) are extracted from the front-end queue
  - Each 'command' contains the 'Target Executor' field which specifies the Executor Interface name
  - Executor interfaces are dynamically instantiated by the APIServer by its name, applying the specified action on DCI
  - Other queue daemons may extract commands from the queue having their own EIs implemented. Targeting for instance other SAGA implementations or even other systems.
  - New EIs can be easily developed just implementing an abstract class (APIServerDaemon)
  - Current available EIs (APIServerDaemon):
    - GridEngine (A core component of the CSGF using JSAGA and targeting: ssh, rOCCI and wms)
    - ToscaIDC (Indigo orchestrator)

Implementation at INDIGO-datacloud

Architecture



fgAPIServer (front-end)

- Available on GIT: https://github.com/indigo-dc/fgAPIServer
- Written in python using Flask microframework http://flask.pocoo.org
- This component listens any FutureGateway API REST call in compliancy with specs defined at: http://docs.csgfapis.apiary.io/#reference
- This service may run as:
  - Standalone service (Normally under a screen section)
    (Good for development environments or small requests traffic rate)
  - WSGI application
    (Suggested for production environments and high requests traffic rate)
    - Different possible configurations: Apache, uWSGI, ...
- The front-end uses a MySQL database to store:
  - Tasks, Applications and Infrastructure with its related data
  - Users/Groups/Roles, Log and Access tokens
  - The task queue

APIServerDaemon (APIServer)

- Available on GIT: https://github.com/indigo-dc/APIServerDaemon
- Servlet that runs a daemon on top of Tomcat application server
  - The Java application was necessary since JSAGA is available only via java language
  - Initially developed to offer a backward compatibility with existing CSGF portal
- Polls over the task queue table
  - Polling timing and other settings can be configured by a dedicated .properties file
  - APIServerDaemon reads tasks requests from the queue, book them as 'to process' and then instruct the correct executor interface for real processing
- Executor interfaces physically interact with DCIs
- Has a consistency check algorithm
  - It re-tries failed requests up to a fixed amount of times. FAILED requests can be reported to the administrator
  - It timely verifies tasks status until their termination; then retrieve task output and updates the DB tables accordingly

FutureGateway

INDIGO - DataCloud

ApiServer Daemon

Executor Interfaces

- GridEngine
- Tosca IDC

Grid and Cloud Engine

Executor Interface name: **gridengine**

- It was the core component of the CSGF
- It uses JSAGA library to address different kind of DCIs
- Usable JSAGA adaptors
  - **EMI/gLite**[*], Globus, **SSH**[*] , **OCCI** [*], UNICORE, Bes Genesis II, Arc, Dirac
- It has an internal auditing system in compliancy with the EGI traceability policies
- Each adaptor requires its own set of application configuration parameters depending on the kind of used adaptor
- Source code available on Git: `https://github.com/csgf/grid-and-cloud-engine/tree/FutureGateway` (Dedicated branch for the FutureGateway)

[*]Tested and used with FutureGateway

TOSCA Orchestrator

Executor Interface name: **ToscaIDC**

- TOSCA is the standard way used inside INDIGO-DataCloud project to create PaaSes resources
- TOSCA provides an endpoint to be contacted via a set of RESTful APIs
- TOSCA Inputs
  - Endpoint accessible through the indigo IAM
    - FutureGayeway bypasses the Token received by the REST call
  - A yaml based template file describing the used PaaS resources
  - A JSON file containing specific values for template variables

```
{ "parameters": { "param_1": "value_1",
                  "param_2": "value_2", … } }
```

- With this adaptor it is possible to simply instantiate a PaaS for long term resources allocation or use the allocated PaaS to execute software
- Long term resources can be destroyed using a PATCH call on task status
- Executed software can perform data IO using ONEDATA

FutureGateway

INDIGO - DataCloud

INDIGO PaaS
TOSCA Orchestrator with ToscaIDC

FutureGateway

ToscaIDC configuration

- ToscaIDC uses the PTV **get-token** service
- In TOMCAT_HOME directory

webapps/APIServerDaemon/WEB-INF/classes/it/infn/ct/ToscaIDC.properties

```
# ToscaIDC properties file

# PTV configuration
# PTV may be required in case FG uses this AAI mechanism

fgapisrv_ptvendpoint= http://localhost:8889
fgapisrv_ptvuser    = tokenver_user
fgapisrv_ptvpass    = *************
```

Portal Token Validation (PTV)

- A service normally provided by a Portal which **verifies** and **get** Tokens
- PTV needs HTTP basic authentication to be contacted
  - Base64 encoded username/password to access the service
  - PTV settings are configurable in API Server Front-end
- PTV usage
  - **Verfiy an incoming token 'check-token'**
    - Accept a POST method with the parameter: token="<token>"
    - If the token is valid returns the AAI user subject value and other info
    - The fgAPIServer uses the returned values to map the portal user with a FG user through a configurable mapping file
  - **Get a new token from a given subject** (AAI User) **'get-token'**
    - Accept a POST method with the parameter: subject="<subject>"
    - Return a new fresh token if the subject is recognized
- Official PTV service inside Indigo-dc project, is LiferayIAM service
  - Liferay7 module available on Git at https://github.com/indigo-dc/LiferayIAM
  - Another PTV implementation are available such MD use case (python)

Config ToscaIDC Application

Stored infrastructure parameters

infra_id	param_id	pname	pvalue	pdesc
7	1	tosca_endpoint	http://localhost:8889/orchestrator/deployments	NULL
7	2	tosca_template	tosca_template.yaml	NULL
7	3	tosca_parameters	params=parameters.json	NULL

- During task submission ToscaIDC expects two input files:
  - The file name expressed by the parameter '**`tosca_template`**' (yaml)
  - The TOSCA parameter file '**`parameters.json`**'
- ToscaIDC will contact the orchestrator via its REST APIs
  - The endpoint is specified in '**`tosca_endpoint`**' parameter

Infrastructure setup:

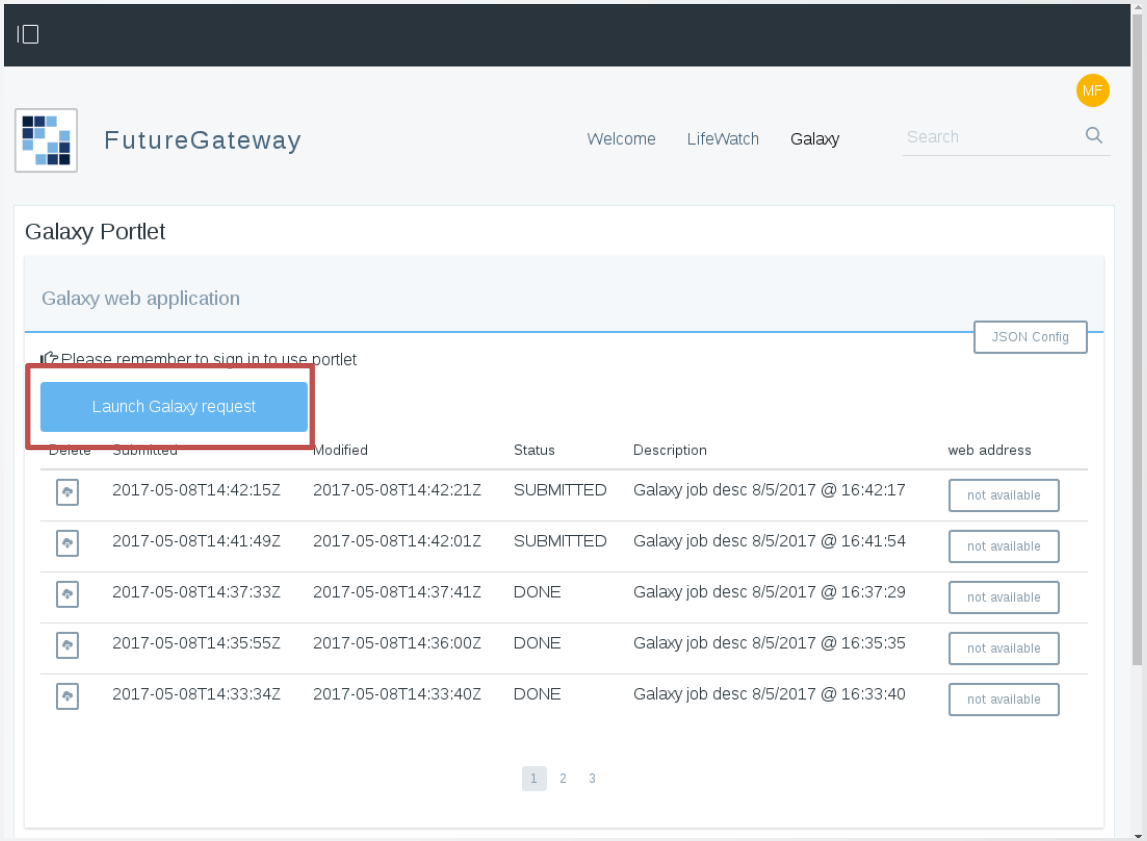
```
{
      "description": "${INFRADESC}",
      "parameters": [
        {
          "name": "tosca_endpoint",
          "value": "${TOSCA_ENDPOINT}"
        },
        {
          "name": "tosca_template",
          "value": "tosca_template.yaml"
        },
        {
          "name": "tosca_parameters",
          "value": "params=parameters.json"
        }
      ],
      "enabled": true,
      "virtual": false,
      "name": "${INFRANAME}"
    }
```

Tasks with TOSCA

- Two kind of tasks
  - PaaS instantiation (Long term availability)
  - PaaS instatiation, application execution and resource deallocation (Single run: Chronos)
- On both cases once PaaS resources are allocated by TOSCA orchestrator the task on FG becomes ‘DONE’
- To free allocated resources (Long term)
  - ToscaIDC implements the **`STATUSCH`** command
  - FG APIs foresee the PATCH method on **`tasks`** specifying a the status ‘**`CANCEL`**’ in parameter: ‘**`status`**’
- Only ToscaIDC EI currently supports the task status change

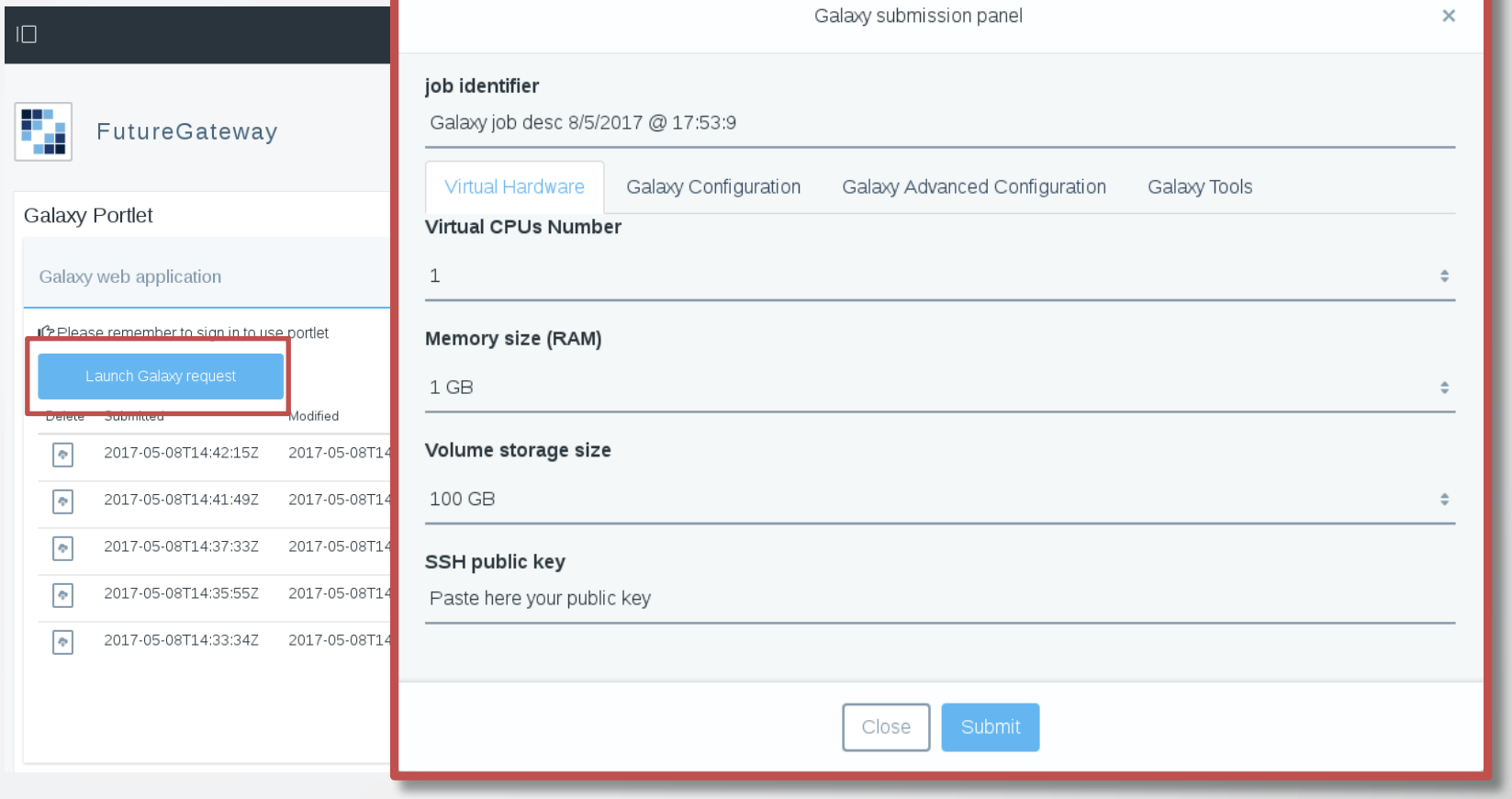
Galaxy use case

- The portlet main interface



Galaxy Submission

- Galaxy use case



Galaxy Submission

JSON Config

default json object can not be changed


```
{
 "version_of_portlet_description": 0.2,
 "tabs": [
  "Virtual Hardware",
  "Galaxy Configuration",
  "Galaxy Advanced Configuration",
  "Galaxy Tools"
 ],
 "parameters": [
  {
   "display": "Virtual CPUs Number",
   "name": "number_cpus",
   "type": "list",
   "value": [
    1,
    2,
    4,
    8,
    16,
    32,
    64
   ],
   "tab": 0
  },
```


Close OK

Galaxy submission panel

ion Galaxy Advanced Configuration Galaxy Tools

Close Submit

Liferay Admin portlet

FutureGateway

INDIGO - DataCloud

Tasks | Applications | Infrastructures

Id	Name
6	LifeWatch
5	GalaxyCluster
4	opn-term
3	kepler-batch
2	SayHello
1	hostname

Tasks | Applications | Infrastructures

Id	Name
5	SSH jobtest@localhost
4	TOSCA @ orchestrator01-indigo.cloud.ba.infn.it
3	hello@eumed
2	sayhello@nebula
1	hello@csgfsdk

Name *

This field is required.

Enabled

Description

Type of application

RESOURCE

Infrastructure

hello@csgfsdk
TOSCA @ orchestrator01-indigo.cloud.ba.infn.it
SSH jobtest@localhost

Files

+

Parameters

+

Save

Infrastructure: 4

- _links
- name
- parameters
  - 0
    - name
      - tosca_endpoint
    - value
      - https://orchestrator01-indigo.cloud.ba.infn.it/orchestrator/deployments
    - description
  - 1
  - 2
- date
- enabled
- id
- virtual
- description

Delete

FutureGateway

New applications

- Use **`$FGLOCATION/gAPIServer/apps/toscaIDCTest`** folder
  - Make a full copy of the folder
  - Customize the **`setup_app.sh`** script and launch it
  - Be sure about the content of TOSCA template yaml and its parameters.json files.
  - Higly suggested to test TOSCA first using curl or Orchent
  - Use **`fgapiserver_ptv.py`** simulator to test APIs without involving the real TOSCA orchestrator
- Use the 'Administration portlet' to automatically setup an application making use of ToscaIDC EI
- Steps related to the application installation are available during the Demo session

Questions

?